Watching subgraphs to improve efficiency in maximum clique search

Pablo San Segundo, Cristobal Tapia, Alvaro Lopez

Abstract. This paper describes a new technique referred to as *watched subgraphs* which improves the performance of BBMC, a leading state of the art exact maximum clique solver (MCP). It is based on watched literals employed by modern SAT solvers for boolean constraint propagation.

In efficient SAT algorithms, a list of clauses is kept for each literal (it is said that the clauses 'watch' the literal) so that only those in the list are checked for constraint propagation when a (watched) literal is assigned during search.

BBMC encodes vertex sets as bit strings, a bit block representing a subset of vertices (and the corresponding induced subgraph) the size of the CPU register word. The paper proposes to watch two subgraphs of critical sets during MCP search to efficiently compute a number of basic operations. Reported results validate the approach as the size and density of problem instances rise, while achieving comparable performance in the general case.

1 Introduction

A *clique* (alias complete graph) is a simple graph with all its vertices pairwise adjacent. The problem of knowing whether a clique subgraph of k vertices exists in a given graph is an NP-complete problem known as *k-clique*. The corresponding optimization problem is the *maximum clique problem* (MCP) whose goal is finding the largest possible clique. MCP is known to have important practical applications related to matching and has been related to many fields such as combinatorial chemistry [1], computer vision [2], global robot localization [3] etc.

In recent years research on MCP has produced a number of very efficient exact algorithms which have improved average performance in more than two orders of magnitude [4-8]. Of these, current leading algorithms are MCS [6] and bit parallel BBMC [7-8]. Specifically, Prosser in a comparison paper [9] reports BBMC as fastest for a number of instances in well known benchmarks. A release version of BBMC for Win-64bit S.O. is publicly available [10].

This paper describes a new idea which improves BBMC performance in large dense graphs while introducing minor overhead in the general case. We refer to the new idea as w*atched subgraphs*. The idea is based in *watched literals* used in modern SAT solvers for boolean constraint propagation as in [11]. In MCP, watched

subgraphs help to improve empty set detection, as well as vertex selection and critical bit mask computations at each step.

The remaining part of the paper is structured as follows: sections 2 and 3 deal with definitions and related work in the field; section 4 presents the new algorithm BBMCW which uses watched sets; section 5 reports empirical results and finally section 6 shows conclusions and summarizes contribution.

2 Preliminaries and notation

A simple undirected graph $G=(V,E)$ consists of a finite set of vertices V and edges E containing pairs of distinct vertices ($E \subseteq VxV$). $N(v)$ refers to the neighbor set of v, i.e. the set of adjacent vertices. For a subset of vertices U, $G(U)$ denotes the subgraph induced by U. Additional standard notation used in the paper for graph invariants include deg(v) for vertex degree (i.e. $|N(v)|$), $\Delta(G)$ for graph degree (i.e. $\max_{v \in V}(\deg(v))$) and $\omega(G)$ for the size of the maximum clique in G.

Vertex coloring is another classical problem in graph theory; its goal is to achieve a (minimum) label assignment (usually referred to as *coloring*) $c(v): V \rightarrow \mathbb{N}$ such that adjacent vertices must all have different labels. Any graph coloring of size n (an *n-coloring*) $C(G)=\{C_1, C_2, \cdots, C_n\}$ partitions V in n disjoint *color classes* C_i (i.e. $v \in C_i \Leftrightarrow c(v)=i$). For a given vertex subset $U \subseteq V$, $C(U)$ or $C(G(U))$ will denote a vertex coloring of U.

Vertex coloring is very much related to efficient MCP search because the size of any feasible coloring $|C(G)|$ is an upper bound on the size of any maximum clique in G, i.e.:

$$|C(G)| \geq \omega(G) \tag{1}$$

3 Reference algorithm and related work

Procedure REF in Table 1 describes the outline of recent efficient MCP algorithms and corresponds roughly to MCQ [4]. Variables used in REF include:

- U: The remaining subgraph hanging from the current node. When a subscript is added (i.e. U_v) it refers to the new remaining child subgraph after expanding v.
- $C(U)$ or C when it is clear from the context: A vertex coloring of U.
- S: The currently growing clique (path of the search).
- S_{max}: The maximum clique found at any moment.

REF enumerates all possible maximal cliques, one per branch. It starts with empty sets S and S_{max}. At each node a new candidate vertex is selected in step 2 and added to S (step 5) until a leaf node is found, when S is a maximal clique in G and is

stored in $S_{max.}$ Each time a new leaf node is reached, S is evaluated to see if it can unseat the current champion in S_{max} (step 7). On backtracking, REF deletes v from S (step 10) and the whole process is repeated until all possible maximal cliques have been enumerated.

U refers to the set of candidate vertices at each node, i.e. those that may enlarge the current clique S in a given step. The new candidate vertex is picked from U (step 2) and the new child subgraph $G(U_v)$ is computed (step 6) by:

$$U_v = U \cap N_U(v) \tag{2}$$

Note that in every node all vertices in U_v are pairwise adjacent to every vertex in S_v and therefore valid candidates to enlarge it.

Table 1. The reference maximum clique algorithm REF.

REF (U, C, S, S_{max})	REFCOL($G(U), C$)
Input: A graph $G(V, E)$	Input: A subgraph $G(U)$
Ouput: A maximum clique in G in S_{max}	Output: A coloring $C(U)$; sorts U by color label
$U \leftarrow G$, $c(v_i) \leftarrow \min\{i, \Delta G\}$, $S, S_{\max} \leftarrow \phi$	
1. **Repeat Until** $U = \phi$	1. sort vertices in U by non-increasing degree
2. | $v \leftarrow \max_{v \in U}(c(v))$	2. SEQ(G, C)
3. | $U \leftarrow U \setminus \{v\}$	3. sort vertices in U by non-increasing color in C
4 | **If** ($\|S\| + c(v) \le \|S_{\max}\|$) **Return**	
5. | $S \leftarrow S \cup \{v\}$	
6. | $U_v \leftarrow U \cap N_U(v)$	
7. | **If** ($U_v = \phi$)	
| | **If**($\|S\| > \|S_{\max}\|$) **Then**	
$\|S_{max}\| \leftarrow \|S\|$	
| | **Return**	
| **endIf**	
8. | REFCOL($G(U_v), C_v$)	
9. | REF ($U_v, C_v, S, S_{\max}$)	
10.| $S \leftarrow S \setminus \{v\}$	
11. **endRepeat**	

A tight upper bound for $\omega(G(U_v))$ is the size of vertex coloring ($|C_v|$) output of procedure REFCOL (table 1, right column). It is used to prune the search space in step 4 of REF (cf. [7] for a detailed explanation). REFCOL uses standard sequential

heuristic coloring procedure SEQ: vertices are selected in a strict order as fixed on input and assigned the smallest possible label consistent with vertices already colored. A key advantage of using approximate coloring for bounding is that the color label $c(v_k)$ of any vertex $v_k \in U_v$ can be used as an upper bound on $\omega(G(W))$ where W is the subgraph with vertices lower than v_k (i.e. $W \subseteq U_v = \{v_1, v_2, \ldots, v_{k-1}\}$) as long as $c(v) \leq c(v_k)\ \forall v \in W$. In practice, REF always picks the candidate vertex with maximum color label to ensure this property (step 2), an important decision heuristic for the domain first described in [4].

3.1 Bit -parallel MCP

An important recent algorithm for exact MCP is BBMC. BBMC employs bit strings to encode the domain so that main operators for child node computation (2) and SEQ coloring are reduced to bitmasks. BBMC also includes a number of subtle improvements such as *class coloring* implemented in the approximate coloring algorithm BBCOL and a fixed initial sorting of vertices by degree (cf. [7-8] for specific detailes).

Notation for bit string encodings used in this paper includes $BS(U)$ for vertex set U, $BS(v)$ for $N(v)$ (the neighbor set of v), and $BS[i]$, the induced subgraph by vertices encoded in the *i-th* bit block of BS. A subindex will be used when the encoded set needs to be made explicit (i.e. $BS_U[i]$).

In all cases the relative position of 1-bits inside the bit strings corresponds with the vertex index in the input graph as fixed initially. The adjacency matrix of the input graph is stored in memory as rows of bit strings $BS(v), \forall v \in V$.

4 Watched subgraphs

This paper proposes to improve the efficiency of BBMC by *watching* two particular subgraphs of critical vertex sets (the ones containing the lowest and highest numbered vertices), and updating them on the fly during search. When this occurs, we also say the vertex set is *watched*.

In the case of MCP, watched critical sets allow for faster computation of the empty set condition for U_v in step 7 in REF as well as for the auxiliary sets used in approximate coloring (cf. [7-8]). Watched sets also allow for more efficient vertex reading from the compact bit strings. Moreover, critical computations for child node generation and coloring are simplified by reducing the range of meaningful bit mask operations.

The size of a watched subgraph is, at most, the register word size of the CPU (typically 64 in modern computers) so that any one of them can be encoded in a single bit block. Whenever a watched subgraph becomes empty, the nearest non empty subgraph becomes watched as a result of an appropriate update operation. The implementation requires two extra indexes (alias *sentinels*) s_l, s_h per watched

set, which point to the watched subgraphs. Notation $BS_U(s_l, s_h)$ is used to indicate that set U is being watched; $BS_U[s_l]$ and $BS_U[s_h]$ refer to the lower and upper subgraph respectively.

Table 2. Basic procedures INIT_WATCH, UPDATE, UPDATE_LOW and IS_EMPTY to operate with watched sets.

INIT_WATCH($BS(s_l, s_h)$)

1. s_l : **If** ($BS \neq \phi$) $s_l \leftarrow$ lowest non empty bitblock; **If** ($BS = \phi$) $s_l \leftarrow -1$
2. s_h : **If** ($BS \neq \phi$) $s_h \leftarrow$ hightest non empty bitblock; **If** ($BS = \phi$) $s_h \leftarrow -2$

UPDATE($BS(s_l, s_h)$)

1. **If** ($s_l == -1 \,||\, s_h == -2$) **Return**
2. **While** ($BS[s_l] == \phi$)
3. | $s_l \leftarrow s_l + 1$
4. | **If** ($s_l > s_h$)
5. | | $s_l \leftarrow -1$, $s_h \leftarrow -2$
6. | | **Return**
7. | **endIf**
8. **endWhile**
9. **While** ($BS[s_h] == \phi$)
10. | $s_h \leftarrow s_h - 1$
11. **endWhile**

UPDATE_LOW($BS(s_l, s_h)$)

1. **If** ($s_l == -1 \,||\, s_h == -2$) **Return**
2. **While** ($BS[s_l] == \phi$)
3. | $s_l \leftarrow s_l + 1$
4. | **If** ($s_l > s_h$)
5. | | $s_l \leftarrow -1$, $s_h \leftarrow -2$
6. | | **Return**
7. | **endIf**
8. **endWhile**

IS_EMPTY($BS(s_l, s_h)$)

1. **If** ($s_l > s_h$) **Return** TRUE **Else Return** FALSE

Table 2 shows the basic procedures implemented to operate with watched sets. Whenever a vertex set needs to be watched, INIT_WATCH is called to initialize the sentinels. Procedure UPDATE will typically be called after a critical operation which changes the set (i.e. child node computation step 6 in REF, or neighbor set removal during coloring (step 6 in BBCOLW, listed in table 3)) and updates the sentinels if required. Note that if both sentinels have the same index the watched set cannot be empty and all its vertices must belong to the pointed subgraph. Note also that UPDATE does not require updating the upper sentinel if empty set condition in step 4 of UPDATE is met. UPDATE_LOW is just a con-

venient case which only updates the lower sentinel. Finally, the empty set condition is evaluated by IS_EMPTY in constant time comparing both sentinels:

$$U = \phi \Leftrightarrow s_l > s_h\,,\ (s_l, s_h) \in BS(U) \tag{3}$$

Table 3 describes the use of watched sets in the new coloring procedure BBCOLW, which substitutes BBCOL in BBMC [cf. 7]. Both sets U and U' are watched so that terminating conditions for both the outer and inner loops are now computed by IS_EMPTY in constant time.

Table 3. The new coloring algorithm which uses watched sets.

BBCOLW ($G(V,E)$, C, L)
Input: A graph G to be colored
Output: $C(G)$, L (vertex set V ordered by color label)

$U \leftarrow V, U' \leftarrow V, L \leftarrow \phi, C \leftarrow \phi, i \leftarrow 1$
//initial step
INIT_WATCH($BS(U')$)
INIT_WATCH($BS(U)$)

1. **Repeat Until** IS_EMPTY($BS(U')$)
2. | **Repeat Until** IS_EMPTY($BS(U)$)
3. | | select the first vertex v from $BS_U[s_l]$
4. | | $C_i \leftarrow C_i \cup \{v\}$ //v is labeled with color i
5. | | $L \leftarrow L \cup \{v\}$ //output vertex list sorted by color
6. | | $U \setminus (\{v\} \cup N(v))$ in range ($BS_U[s_l], BS_U[s_h]$)
7. | | UPDATE_LOW($BS(U)$)
8. | | $U' \leftarrow U \setminus \{v\}$ //removes colored vertex
9. | **endRepeat**
10. | UPDATE($BS(U')$)
11. | $U \leftarrow U'$, $(s_l, s_h) \leftarrow (s_l', s_h')$ //copies also sentinels
12. | $i \leftarrow i+1$ //new color
13. **endRepeat**

5 Experiments

This section reports tests undertaken to evaluate the proposed watched set strategy (implemented as BBMCW) w.r.t. state of the art BBMC. The computer employed for the report is an Intel(R) Core(TM) i7 CPU 950 @ 3.07GHz with a 64-bit Windows

O.S. and 6GB of RAM. It is the same as the one used in [8]; machine times for *dfmax* algorithm are available there (c.f. appendix section). In all experiments there was a time limit of 3600s and tests where averaged over 10 runs. We note that BBMC has been slightly modified to facilitate deployment and management w.r.t. versions described in [7-8] so times reported may differ slightly from those published elsewhere. We also note that only a small representative subset of all the experiments are published because of space constraints.

Table 4 reports results for a set of uniform random graphs (left) as well as a subset of the well known DIMACS graph benchmark [12] (right). Columns n, p refer to size and density of each graph; time is measured in seconds. The table shows that the impact of watched sets improves with size and density up to a 40% approx. In the random case, this is more acute for $n \geq 1000$. In the DIMACS graphs the differences in performance are smaller since all the graphs have less than 1000 nodes except two cases in the *phat* family. It is here that BBMCW performs best.

Table 4. User times for BBMC and BBMCW over a set of uniform random graphs and a subset of DIMACS graphs [12]. In bold best times for each test.

n	p	BBMC	BBMCW
300	0.50	**0.034**	0.036
300	0.60	**0.290**	0.297
300	0.70	**3.977**	4.148
500	0.40	0.117	**0.106**
500	0.50	0.937	**0.867**
500	0.60	15.599	**14.593**
1000	0.30	0.511	**0.442**
1000	0.40	6.696	**5.756**
1000	0.50	156.732	**136.255**
3000	0.10	0.371	**0.248**
3000	0.20	8.345	**5.819**
5000	0.10	2.894	**1.822**
5000	0.20	149.517	**104.387**
1000	0.10	57.356	**35.702**
1500	0.10	341.670	**226.890**

name	n	p	BBMC	BBMCW
brock200_1	200	0.745	0.249	**0.240**
brock200_2	200	0.496	0.002	0.002
brock200_3	200	0.605	0.009	0.009
brock200_4	200	0.658	0.043	**0.041**
phat700-2	700	0.498	2.777	**1.716**
phat1000-2	1000	0.49	150.610	**79.467**
phat1500-1	1500	0.253	2.605	**1.591**
hamming10-2	1024	0.99	0.019	**0.014**
keller4	171	0.649	0.008	**0.006**
san200_0.7_2	200	0.7	0.001	0.001
san200_0.9_1	200	0.9	**0.012**	0.015
san200_0.9_2	200	0.9	**0.034**	0.038
sanr200_0.7	200	0.702	0.097	**0.094**
sanr400_0.5	400	0.501	0.237	**0.216**
sanr400_0.7	400	0.7	71.685	**65.495**

6 Conclusions

This paper describes a novel idea of *watched subgraphs* to improve the performance of BBMC, a leading bit parallel solver for exact maximum clique. The idea is inspired in watched literals employed in efficient SAT solvers for Boolean constraint propagation. Reported results show that the efficiency of BBMC with watched sets improves as size and density of input graphs increase. Moreover, watched sets show comparable running times w.r.t. previous BBMC in the general case.

7 Acknowledgments

This work is funded by the Spanish Ministry of Science and Technology (ARABOT: DPI 2010-21247-C02-01) and supervised by CACSA whose kindness we gratefully acknowledge.

8 References

1. Bahadur, D.K.C., Akutsu, T., Tomita, E., Seki, T., Fujijama, A.; Point matching under non-uniform distortions and protein side chain packing based on efficient maximum clique algorithms. Genome Inform. 13 (2006), 143-152.
2. Hotta, K., Tomita, E., Takahashi, H.; *A view invariant human FACE detection method based on maximum cliques*. Trans. IPSJ, 44, SIG14 (TOM9) (2003), 57-70.
3. San Segundo, P., Rodríguez-Losada, D., Matía, F., Galán, R.; *Fast exact feature based data correspondence search with an efficient bit-parallel MCP solver*. Applied Intelligence, 32(3) (2010) 311-329.
4. Tomita E., Seki, T.; *An efficient branch and bound algorithm for finding a maximum clique*. Proc. Discrete Mathematics and Theoretical Computer Science. LNCS 2731 (2003) 278-289.
5. Konc, J., Janežič, D.; *An improved branch and bound algorithm for the maximum clique problem*. MATCH Commun. Math. Comput. Chem. 58 (2007) 569-590.
6. Tomita E., Sutani Y., Higashi T., Takahashi S., Wakatsuki M.: *A simple and faster branch-and-bound algorithm for finding a maximum clique*. In Lecture Notes in Computer Science, 5942. Edited by Rahman MS, Fujita S. (2010) 191-203.
7. San Segundo, P. Rodriguez-Losada, D., Jimenez, A. *An exact bit-parallel algorithm for the maximum clique problem*. Computers & Operations Research 38(2) (2011) 571-581.
8. San Segundo, P., Matia, F, Rodriguez-Losada, D, Hernando, M. *An improved bit parallel exact maximum clique algorithm*. Optimization Letters (2011) Online DOI: 10.1007/s11590-011-0431-y.
9. Prosser, P. *Exact Algorithms for Maximum Clique: A Computational Study*. Algorithms 5(4) (2012) 545-587.
10. BBMC release for 64 bit Win O.S. is available at: http://www.intelligentcontrol.es/arabot/sites/default/files/frontpage/bbmc1.0.zip
11. Moskewicz, M., Madigan C., Zhao, Y., Zhang, L. and Malik, S. *Chaff: engineering an efficient SAT solver*. In XXXVIII Proc. annual Design Automation Conference (DAC '01). ACM, New York (2001) 530-535.
12. Johnson, D.S. Trick, M.A (eds.); Cliques, coloring and Satisfiability. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 26. American Mathematical Society, Providence (1996).